IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s):	Marc Tremblay		
Title:	PIPELINED PROCESSOR WITH MULTI-CYCLE GROUPING FOR INSTRUCTION DISPATCH (AS AMENDED)		
Application No.:	09/583,097	Filed:	August 2, 1999
Examiner:	David J. Huisman	Group Art Unit:	2183
Atty. Docket No.:	004-1391-1	Confirmation No.:	7166

June 6, 2005

Mail Stop Appeal Briefs - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF (37 C.F.R. § 41.37)

This brief is in furtherance of the Notice of Appeal, filed on March 2, 2005 and received by the Office on March 7, 2005. A Petition for Extension of Time accompanies this Brief, extending the period for filing until June 7, 2005. The fee required under 37 C.F.R. § 41.20(b)(2) is provided in the accompanying Transmittal.

REAL PARTY IN INTEREST

The real party in interest in this appeal is Sun Microsystems, Inc., the assignee of record, as evidenced by the assignment recorded at Reel/Frame 008052/0437.

RELATED APPEALS AND INTERFERENCES

Applicant is unaware of any prior or pending appeal, interference or judicial proceeding, which may be related to, directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

STATUS OF CLAIMS

Claims 9-36 are pending. Claims 9-36 stand as rejected. Rejected claims 9-34 are the subject of this appeal.

STATUS OF AMENDMENTS

An amendment was filed, 7 February 2005, subsequent to the final rejection. That amendment has been entered for purposes of appeal as indicated by the Advisory Action mailed 17 February 2005 and apparently disposes of various claim informalities objected to by the Examiner. A second after-final amendment accompanies this Brief. In that second after-final amendment, Applicant seeks to cancel claims 35 and 36 and thereby simplify issues for appeal. The above-mentioned Status of Claims reflects the entry of the first after-final amendment, but not the second.

SUMMARY OF CLAIMED SUBJECT MATTER

As a general matter, realizations of the presently claimed subject matter provide a central processing unit that includes a grouping logic circuit for determining a set of instructions simultaneously dispatchable in a processor cycle. The grouping logic includes a number of pipeline stages, such that resource allocation and data dependency checks can be performed over a number of processor cycles, rather than as traditionally performed, within the span of a single processor cycle immediately prior to dispatch. See page 2, line 36 through page 3, line 33. As a result, techniques of the present invention allow for simultaneous dispatch of a large number of instructions, while avoiding or managing increases in complexity that had previously resulted in grouping logic circuits becoming a limitation on improvements in processor cycle time. In contrast, prior art techniques (including those relied upon in the present final rejection) tend to perform data dependency checking and resource allocation (if applicable) in within the span of a single processor cycle thereby becoming a critical path and placing a practical lower bound on cycle time. See page 7, lines 10-26.

Turning now to the appealed claims, independent claim 9 is directed to a superscalar processor that, for a given instruction instance, performs, over plural execution cycles of the superscalar processor, instruction grouping for dispatch. *See* Applicant's functional description

of a grouping logic circuit, page 3, lines 25-28, page 7, lines 24-31 (with reference to Figure 1, element 109). The instruction grouping for dispatch takes the plural execution cycles to complete and includes both intra-group and inter-group dependency checking. Applicant's description of a four-stage grouping logic pipeline 200 at page 11, lines 14-32 (and with reference to Figure 2) is illustrative.

Dependent claim 13 (which is separately argued) further recites that the intra-group dependency checking spans at least two of the plural execution cycles. *See* page 11, lines 5-9.

Dependent claim 15 (which is also separately argued) further recites that data dependency and resource allocation checks in earlier pipeline stages of instruction grouping are based, at least in part, on a predicted subsequent state of the superscalar processor. Subsequent (i.e., future) state $S(t+\tau)$ and related processes for predicting future state are described beginning at page 7, line 26 and continuing through page 10, line 24.

Independent claim 17 is directed to a processor comprising plural functional units and grouping logic coupled to the functional units. The plural functional units execute instructions in respective numbers of processor cycles. *See* Figure 1 and description thereof beginning at page 4, line 17 for discussion of illustrative functional units and pipelining thereof. The grouping logic is itself pipelined to compute, over plural cycles, T, of the processor, a future state, S(t+T), of the processor based on a prior state, S(t), of the processor and based thereon to select a group of instructions from a program sequence thereof for dispatch to the functional units. The future state computing takes the plural cycles to complete. Applicant describes the process for predicting future state, $S(t+\tau)$, first with examples of $\tau = 1$ and then, by extension, for values of $\tau = T$, $T > 1$. *See* page 7, line 26 through page 10, line 24; see also, description of an illustrative four stage (T=4) pipelined grouping logic.

Dependent claim 18 (which is separately argued) further recites that the intra-group dependency checking by the pipelined grouping logic spans two or more of the T cycles. *See* page 11, lines 5-9.

Independent claim 26 is directed to a method of operating a processor, the method comprises identifying successive groups of instructions for dispatch to respective ones of plural execution units of the processor, performing dependency checking during plural pipelined execution cycles of the processor, and dispatching instructions. The dependency checking is (i) amongst instructions of a later one of the groups (*see* at least stage 201 described at page 11, lines 15-17) and (ii) between the instructions of the later group and instructions of a preceding one of the groups (*see e.g.*, stage 203 described page 11, lines 19-23). The dependency checking takes the plural pipelined execution cycles to complete. Instructions of the later group are dispatched only after all instructions of the preceding group have been dispatched. *See e.g.*, page 11, lines 1-3 and lines 32-33.

Dependent claim 28 (which is also separately argued) further recites that intra-group dependency checking and within-group resource allocation are performed in successive processor cycles by pipelined grouping logic. *See e.g.*, stages 201 and 202 of illustrative four-stage, pipelined grouping logic, which are described at page 11, lines 15-19 with reference to Figure 2.

Independent claim 31 is directed to a method comprising, in a first cycle of processor execution, identifying plural candidate instructions for an instruction group (see page 10, lines 34-35). The method further including, beginning, in a subsequent cycle of processor execution, intra-group dependency checking as amongst instructions of the instruction group (*see e.g.*, page 10, lines 35-36, page 11 lines 5-6, and description (with reference to Figure 2) of first stage 201 at page 11, lines 15-17). The method further includes checking, in a cycle of processor execution prior to dispatch of any instruction from the instruction group, non-deterministic conditions (*see e.g.*, description of stage 204 at page 11, lines 29-31 and related illustration of possible store buffer 105 and/or load buffer 104 conditions at page 9, line14 through page 10, line 7). The method further includes initiating, in a cycle of processor execution prior to the non-deterministic dependency condition checking, inter-group dependency checking between instructions of the instruction group and instructions of one or more prior instruction groups (*see e.g.*, description of stage 203 at page 11, lines 19-23). For instruction instances of the instruction group, dependency checking, which includes the intra-group dependency checking and the inter-

group dependency checking, takes plural of the cycles of processor execution to complete (*e.g.*, cycles corresponding to at least stages 201 and 203 described at page 11, lines 15-29).

Dependent claim 34 (which is also separately argued) further recites that the non-deterministic condition checking is performed aggressively, computing dispatch conditions for at least two alternatives including no change in condition and condition resolution and that a control signal is selective for a particular one of the computed dispatch conditions. *See* page 9, line 31 through page 10, line 7.

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Ground I: The rejection of claims 9-34 under 35 U.S.C. § 102(b) as being anticipated by U.S. Patent 5,193,167 to Sites et al. (hereafter *Sites*).

ARGUMENT

Claims 9–34 were rejected under 35 U.S.C. 102(b) as being anticipated by Sites et al., U.S. Patent No. 5,193,167 (hereinafter *Sites*). In rejecting the pending claims, the Office simply misinterprets certain aspects of *Sites'* pipeline. In particular, the Office fails to properly identify any express or inherent disclosure of pipeline stages during which intra-group dependency checking and inter-group dependency checking are performed in accordance with the claim language.

Applicant respectfully suggests that five (5) errors of interpretation are significant:

(1) The Office apparently construes *Sites'* first four pipeline stages, i.e., fetch (S0), swap (S1), decode (S2) and register access/issue (S3) as "instruction grouping for dispatch, including both intra-group and inter-group dependency checking." *See* Final Action ¶12. In particular, the Office states that: "over multiple cycles (S0-S3) plus any stall cycles, dependencies and resource constraints are checked."

The Office's interpretation of Sites' initial pipeline stages through decode (i.e., fetch (S0), swap (S1) and decode (S2)) as dependency checking ignores the simple fact that decode (e.g., of register addresses) must have been completed to allow for checking of dependencies (intra-group or inter-group) between instructions. Accordingly, dependency checking is performed in Sites, if at all, after decode (S2) in register access/issue (S3) stage. Of note, the register access/issue (S3) stage is a single-cycle stage.

(2) Related to (1), the Office's analysis seems to suggest (and perhaps relies upon reasoning that) a pipeline stall acts to transform a single-cycle operation into a multiple-cycle operation. *See* Final Action ¶12.

To the contrary, an operation that is performed in one cycle is still performed in one cycle whether or not, after performance of that operation, the pipeline is stalled for one or more subsequent cycles.

(3) The Office further construes *Sites'* first two pipeline stages, i.e., fetch (S0) and swap (S1), as intra-group dependency checking that spans multiple cycles. *See* Final Action, ¶16. To be specific, the Office states that: "fetching is considered a portion of intra-group dependency checking because if instructions aren't fetched, then they cannot be checked for dependencies.

Applicants respectfully disagree.

By the Office's reasoning: examination of a patent application must necessarily include both invention and application drafting since examination cannot be performed unless an invention is made and an application prepared. The Office's reasoning here is simply unsustainable.

(4) Next, and perhaps fundamentally, the Office assumes that Sites' swap stage (S1) constitutes intra-group dependency checking. *See* Final Action throughout, but particularly ¶¶16, 17.

No intra-group dependencies are checked at Sites' swap stage (S1). Instead, Sites' swap stage (S1) purports to determine (based on instruction type, not based on any dependency) whether two fetched instructions must be issued separately (i.e., in two distinct issue groups) or if the two fetched instructions may be issued at the same time, subject to dependency checking (presumably intra-group and inter-group dependency checking) performed at register access/issue stage (S3) after register addresses are decoded in decode stage (S2). In short, it is rather odd for the Office to take the position that intra-group dependency checking is performed in a stage that precedes the decoding of register addresses that would presumably be employed in the dependency checking.

(5) Finally, with respect to claim language relating to evaluation of "non-deterministic conditions," the Office leverages a definition of "nondeterminism" (Final Action, ¶¶19, 23, 33, 34) in a way that is simply absurd and contrary to accepted meaning.

While it is true that non-determinism is generally understood to be a property of a computation that may have more than one result, the Office neglects the additional and critical point that a nondeterministic computation is generally one that <u>for a given set of inputs</u> may have more than one result. Thus, addition is not non-deterministic because 2+2 and 2+3 yield different results. Similarly, intra-group dependency checking is not non-deterministic because one group of instructions and register targets exhibits a dependency while another group of instructions and register targets does not.

> ***Accordingly, in the context of the present application, a non-deterministic condition is one that, based on information available at a given time, cannot be precisely determined. The specification gives examples of future load buffer or store buffer state that (as implemented) are not precisely determinable.***
>
> ***Applicant respectfully suggests that the Office has misapplied its definition in a way that is nonsensical. To aid the Office, Applicants suggest the following definition may be less susceptible to misapplication:*** NONDETERMINISTIC- permitting more than one choice of next move at some step in a computation ***(From Algorithms and Theory of Computation Handbook, page 24-19, Copyright ©1999 by CRC Press LLC. Appearing in the Dictionary of Computer Science, Engineering and Technology, Copyright © 2000 CRC Press LLC; see also http://www.nist.gov/dads/HTML/nondeterministic.html)***

Now, based on the preceding, we take up the Office's rejection of specific claims. For ease of description, we begin with claim 26 *et seq.*

Claims 26, 27, 29 and 30

With respect to claims 26, 27, 29 and 30, Applicants respectfully submit that *Sites* fails to teach at least the following element(s) of the claims:

> performing, during plural pipelined execution cycles of the processor, dependency checking amongst instructions of a later one of the groups and between the instructions of the later group and instructions of a preceding one of the groups
> ...
> wherein the performed dependency checking takes the plural pipelined execution cycles to complete.

Claim 26 (emphasis added).

Contrary to the Office's position, dependency checking in *Sites* is not performed over plural pipelined execution cycles, but rather, is performed conventionally, during a single cycle, just prior to issue, in the fourth stage (S3) of *Sites* pipeline. In *Sites*, four pipeline stages are employed prior to issue but, as described below, it is incorrect to interpret any of the first three stages (S0, S1 or S3) as "dependency checking." Instead, a first stage (S0) is the fetch stage, a second stage (S1) is the swap stage, a third stage (S2) is the decode stage and a fourth stage (S3) is the register file access and issue stage. Sites, col. 10, lines 26-47.

In this regard, the Office's reliance on column 6, lines 22-26 (*see* Final Action, ¶ 29(b)) to dispose of the above-quoted limitation is misplaced. First, the relied upon disclosure does not concern "dependency checking". Second, even with respect to resource availability, the Office's reliance is inconsistent with the full passage. Indeed, the more complete quote from *Sites* (provided below) drives home the point that *Sites'* check for availability of required resources is performed in the stage immediately prior to supply of instructions and register, *i.e.*, after decode (stage s2) in issue stage s3:

> The instruction unit 18 includes an instruction cache 21 which stores perhaps 8 Kbytes of instruction stream data, and a quadword (two instructions) of this instruction stream data is loaded to an instruction register 22 in each cycle where the pipeline advances. The instruction unit 18, in a preferred embodiment, decodes two instructions in parallel in decoders 23 and 24, then checks that the required resources are available for both instructions by check circuitry 25. If resources are available and dual issue is possible then both instructions may be issued by applying register addresses on busses 26 and 27 and control bits on microcontrol busses 28 and 29 to the appropriate elements in the CPU 10.

Sites, col. 6, lines 18-26 (emphasis added).

The Office provides no real factual basis for it's next supposition (*see* Final Action, ¶¶ 29(b), 29(d)) that "dependencies and resources are checked [over multiple cycles (stages S0-S3)]." Indeed, with particular respect to the claim limitation at hand (dependency checking performed over plural pipelined execution cycles), the Office's supposition is simply inconsistent with clear implication of *Sites'* disclosure:

> The third stage S2 is the decode stage, during which the two instructions are decoded in the decoders 23 and 24 to produce the control signals 28 and 29 and register addresses 26 and 27.

Sites, col. 10, lines 41-44 (emphasis added).

Since information regarding of operations and register addresses would necessarily be employed in any dependency checking in *Sites* to identify the source and target conflicts between instructions, it follows that dependency checking in *Sites*, must be performed, if at all, in stage

S3 after such register addresses and operations are decoded (in stage S2). Indeed, the above-quoted material suggests exactly that. Despite this, the Office misinterprets *Sites*, stating that dependencies [and resources] are checked "over multiple cycles (stages S0-S3) plus any stall cycles." *See* Final Action, ¶¶ 29 (b, d). Leaving aside (for the moment the issue of stalls), this interpretation of *Sites* is simply unsustainable as dependency checking in *Sites* pipeline must occur after decode.

Regarding stalls, it is unclear from the Advisory Action whether the Office continues to rely on a stall cycle argument; however, assuming (for the sake of completeness) that the Office continues to rely on such reasoning (*see* Final Action ¶12), Applicant respectfully submits that an operation that is performed in one cycle is still performed in one cycle whether or not, after performance of that operation, the pipeline is stalled for one or more subsequent cycles.

Anticipation under 35 U.S.C. § 102 requires that each and every element of the claimed invention be disclosed in a single prior art reference. Minnesota Mining & Mfg. Co. v. Johnson & Johnson Orthopaedics Inc., 976 F.2d 1559, 1556, 24 U.S.P.Q.2d 1321, 1326 (Fed. Cir. 1992) (To establish anticipation a party "must show that each element of the claim in issue is found, either expressly or under principles of inherency, in a single prior art reference ...."). A prior art reference anticipates a claim only if the reference discloses, either expressly or inherently, every limitation of the claim. *See also*, Verdegaal Bros., Inc. v. Union Oil Co., 814 F.2d 628, 631, 2 U.S.P.Q.2d 1051, 1053 (Fed. Cir. 1987). "[A]bsence from the reference of any claimed element negates anticipation." Kloster Speedsteel AB v. Crucible, Inc., 793 F.2d 1565, 1571, 230 U.S.P.Q. 81, 84 (Fed. Cir. 1986).

As shown above, *Sites* fails to teach each and every limitation of claim 26. In particular, *Sites* fails to disclose dependency checking (as claimed) performed over plural pipelined execution cycles. As a result, claim 26 and those dependent therefrom (claims 27-30) are patentable over *Sites*.

Claim 28

With respect to claim 28, the Office asserts that *Sites* has further "taught that intra-group dependency checking and within group resource allocation

are performed in successive processor cycles," apparently reasoning that a single instruction (of pair) issuance scenario might result in performance (again, apparently in a 2nd pass) of within-group resource allocation for the remaining un-issued instruction of the pair.

Applicant respectfully notes that *Sites* does not disclose this hypothecated 2nd pass within-group resource allocation. Indeed, other than indicating that the instruction unit stalls ("If only the first of a pair of instructions issues (from the decoder 23), the instruction unit 18 does not advance another instruction into the instruction register 22") (col. 6, lines 34-37), *Sites* is silent on pipeline behavior in a single instruction (of pair) issuance scenario. Furthermore, a 2nd pass within-group resource allocation operation does not follow from that which *Sites* does disclose. More importantly, such a mode of operation does not really make sense in the *Sites'* dual-issue architecture since once a single instruction (of the pair) issues, only one instruction remains and no role for "within-group [of one] resource allocation" is apparent. Indeed, the Office's interpretation (in effect) construes the claim limitation as a nullity, since there is no "within-group resource allocation" to be performed amongst members of a group that consists of a single remaining valid instruction available to be issued.

Under principles of inherency, when a reference is silent about an asserted inherent characteristic, it must be clear that the missing descriptive matter is *necessarily present* in the thing described in the reference. *Continental Can Co. v. Monsanto Co.*, 948 F.2d 1264, 1268, 20 U.S.P.Q.2d 1746, 1749 (Fed. Cir. 1991). Furthermore, inherency may not be established by probabilities or possibilities. The mere fact that a thing may result from a given set of circumstances is not sufficient. *In re Robertson*, 169 F.3d 743, 745, 49 U.S.P.Q.2d 1949, 1950-51 (Fed. Cir. 1999); *Continental Can*, 948 F.2d at 1269, 20 U.S.P.Q.2d at 1749 (quoting *In re Oelrich*, 666 F.2d 578, 581, 212 U.S.P.Q. 323, 326 (CCPA 1981).

In the present case, the hypothecated successive-cycle performance of intra-group dependency checking and within-group resource allocation is not *necessarily present* in *Sites* and cannot be said to necessarily follow from *Sites'* disclosure. Indeed, it is not really plausible (or probable) that, after issuing one (of two) instructions, the processor disclosed by *Sites* would

necessarily repeat "within-group resource allocation" for a "group" that includes but one unissued instruction.

In short, *Sites* does not disclose or suggest, expressly or inherently, the additional limitation(s) of claim 28. Therefore, in addition to the reasons previously given, claim 28 is allowable for at least this reason as well.

Claims 31-33

With respect to claims 31, 32 and 33, Applicants respectfully submit that *Sites* fails to teach at least the following element(s) of the claims:

> ... for instruction instances of the instruction group, dependency checking, which includes the intra-group dependency checking and the inter-group dependency checking, takes plural of the cycles of processor execution to complete.

Claim 31 (emphasis added).

In rejecting the claims, the Office conveniently reads the word "dependency" out of the claim. The Office apparently does so to avoid the problem that, in *Sites* design, "intra-group dependency checking" and "inter-group dependency checking" must be performed, if at all, in one and only one pipeline stage, namely stage S3, after register addresses are decoded, to even allow an evaluation of whether a dependency exists. *See* Application, Background, p. 1, line 21 through p. 2, line 6; Summary, p. 3, lines for usages of the term dependency to describe *data* dependency as contrasted with mere resource availability.

Leaving aside for the moment,[1] whether it is proper to construe *Sites* swap stage S1 as performing an "intra-group check" (as distinguished from the "intra-group dependency

---

[1] Applicant specifically reserves the issue of whether, properly construed, *Sites* should be understood to enablingly disclose any intra-group check at swap stage S1, noting that, while the Examiner chooses to rely on a portion of the statement:

> The second stage S1 is the swap stage, during which the fetched instructions are evaluated by the circuit 25 to see if they can be issued at the same time
>
> (Sites, col. 10, lines 38-41),
>
> the weight of the description as a whole (including the express language of the relied upon statement) establishes that dual issue decision making is performed by circuit 25, which necessarily places such

checking" actually recited in claim 31), it is clear that whatever check may even arguably be hinted at for swap stage S1, it is not a dependency check but rather a resource check. For this reason alone, claim 31 and those dependent therefrom (claims 32-34) are allowable over *Sites*.

Finally, it is notable that the Office asserts elsewhere in its rejection of claim 31 (*See* Final Action ¶34(b)) that an "intra-group check" in *Sites* is performed "`after instructions are decoded`" (i.e., in stage S3 following decode stage S2 and consistent with the interpretation urged by Applicants). The Office cannot construe "intra-group dependency check" one way (i.e., as a stage S3 operation) to dispose of a first limitation, then another inconsistent way (i.e., as a stage S0-S2 operation) to dispose of another limitation of the same claim. For this reason as well, claim 31 and those dependent therefrom (claims 32-34) are allowable over *Sites*.

Claim 34

In rejecting claim 34, the Office simply ignores the clear limitation of the claim which requires:

> … the non-deterministic condition checking is performed aggressively, computing dispatch conditions for at least two alternatives including no change in condition and condition resolution.

In short, Applicant's design computes outcomes for both of at least two alternative possibilities of a non-deterministic condition whose result cannot be precisely determined based on information available at the time. In this way, on either result, a dispatch condition has been computed. There is simply no disclosure or suggestion of such an approach in *Sites*.

The Office's error may trace to its prior interpretation of claim limitations (in claim 31 from which 34 depends, and elsewhere) involving "non-deterministic condition" checks as an effective nullity. Applicant has previously pointed out the Office's errant extrapolation of a definition for "non-determinism" and has provided a more correct and substantive definition in the response filed 7 February 2005. That more correct and substantive definition is repeated above.

---

performance in stage S3, since circuit 25 follows decode logic (and associated stage S2) in the pipeline flow. In short, the Office has ignored the clear statement(s) attributing dual issue decision making to circuit 25 (stage S3) in favor of one inconsistent (and arguably errant) statement that better supports its theory of anticipation.

In any case, there is simply no disclosure or suggestion, in *Sites*, of an approach that involves computing dispatch conditions for both of two alternative results of a non-deterministic condition check. Therefore, in addition to the reasons previously given, claim 34 is allowable for at least this reason as well.

Claims 9-12, and 14

With regard to claim 9, Applicant respectfully submits that *Sites* fails to teach performance over plural execution cycles of instruction grouping for dispatch (which includes both intra-group and inter-group dependency checking). The Office takes the position that "[dependencies and resource constraints are checked] over multiple cycles (stages S0-S3) plus any stall cycles." Final Action, ¶12.

As described above, the Office misinterprets *Sites* when it so construes *Sites'* first four pipeline stages, i.e., fetch (S0), swap (S1), decode (S2) and register access/issue (S3). Properly construed, *Sites* teaches a single stage, i.e., register access/issue (S3), during which instruction dependency checking is performed after decode.

As before, the Office's interpretation of *Sites* is simply inconsistent with clear implication of *Sites'* disclosure:

> The third stage S2 is the decode stage, during which the two instructions are decoded in the decoders 23 and 24 to produce the control signals 28 and 29 and register addresses 26 and 27.

Sites, col. 10, lines 41-44 (emphasis added).

In particular, the Office's interpretation of Sites' initial pipeline stages through decode (i.e., fetch (S0), swap (S1) and decode (S2)) as dependency checking ignores the simple fact that decode (e.g., of register addresses) must have been completed in *Sites* design to allow for checking of dependencies (intra-group or inter-group) between instructions.

At best, *Sites* is non-enabling for the proposition asserted. At worst, *Sites* (properly construed) is inconsistent the proposition asserted.

Since information regarding of operations and register addresses would necessarily be employed in *Sites* design in any dependency checking to identify the source and target conflicts between instructions, it follows that dependency checking in *Sites*, must be performed, if at all, in stage S3 after such register addresses and operations are decoded (in stage S2). Indeed, the above-quoted material suggests exactly that. Accordingly, dependency checking is performed in Sites, if at all, after decode (S2) in register access/issue (S3) stage, which is a single-cycle stage. Dependency checking is not performed over plural cycles in *Sites*. Claim 9 and those dependent therefrom (claims 10-16) are allowable for at least this reason.

Claim 13

With regard to claim 13, Applicant respectfully submits that no operation of *Sites* properly construable as "`intra-group dependency checking`" spans at least two execution cycles, as claimed. As a preliminary matter, Applicant notes that (as previously argued above), *Sites* swap stage (S1) does not constitute intra-group dependency checking. However, even assuming *arguendo* the Office's faulty premise, it is clear and unambiguous that the preceding stage (fetch stage S0) has nothing at all to do with dependency checking.

Nonetheless, the Office construes *Sites'* first two pipeline stages, i.e., fetch (S0) and swap (S1), as intra-group dependency checking that spans multiple cycles. *See* Final Action, ¶16. To be specific, the Office states that: "`fetching is considered a portion of intra-group dependency checking because if instructions aren't fetched, then they cannot be checked for dependencies.`" Without acquiescing in the faulty premise that swap stage (S1) constitutes intra-group dependency checking, Applicant's note that the Office's attribution of dependency checking to the preceding stage S0 is simply unsupportable on its face. Applicant respectfully requests that the rejection be withdrawn. Therefore, in addition to the reasons previously given, claim 15 is allowable for at least this reason as well.

Claim 15

With regard to claim 15, Applicant respectfully submits that *Sites* fails to disclose or suggest:

> ... data dependency and resource allocation checks in earlier pipeline stages of instruction grouping [that] are based, at least in part, on a predicted subsequent state of the superscalar processor.

In rejecting claim 15, the Office relies on disclosure of *Sites* related to branch prediction, not predicted subsequent state. In particular, while *Sites* processor may actually execute instructions based on predicted taken (or untaken) branch, any data dependency checking and/or resource allocation checks performed are based on an *actual, then-current state* of *Sites* processor (executing along a path of execution that is predicted to be correct). Actual, then-current state of a processor (along a predicted path of execution) is not the same as a predicted subsequent state of the processor. *Sites* does not disclose or suggest the subject matter claimed. Therefore, in addition to the reasons previously given, claim 15 is allowable for at least this reason as well.

Claims 17, 19 and 22-25

With regard to claim 18, Applicant respectfully notes that *Sites*[2] does not disclose or suggest:

> grouping logic coupled to the functional units and pipelined to compute, over plural cycles, T, of the processor, a future state, S(t+T), of the processor based on a prior state, S(t), of the processor and based thereon to select a group of instructions from a program sequence thereof for dispatch to the functional units, wherein the future state computing takes the plural cycles to complete.

In this regard, the Office apparently misinterprets *Sites* initial pipeline stages. In Sites, three stages (fetch S0, swap S1 and decode S2) lead up to a fourth stage (register access/issue S3) in which an issue decision (e.g., a future state S(t+1)) is made (calculated in a single cycle) based on the processor state that then exists (e.g., the then current state S(t)). Operations performed in preceding stages S0, S1 and S2 cannot be properly interpreted as computing a future state of *Sites* processor based on then current state. In particular,

1. a fetch performed at stage S0 does not compute a future state of the processor (4-cycles ahead) based on state of the processor in the then-current cycle;

[2] While the Office purports to reject over "Tremblay" (an apparent reference to Applicant himself), the undersigned presumes that this is a typo and that the rejection is intended to be over *Sites*.

2. a swap performed at stage S1 does not compute a future state of the processor (3-cycles ahead) based on state of the processor in the then-current cycle; and

3. a decode performed at stage S2 does not compute a future state of the processor (2-cycles ahead) based on state of the processor in the then-current cycle.

Instead, all future state computing in *Sites* is performed (1-cycle ahead) in a single-cycle register access/issue stage S3 based on state of the processor in the then-current cycle. Accordingly, Sites does not disclose or suggest the invention recited in claim 17.

Perhaps most instructive in diagnosing the Office's faulty reasoning is its conclusion: "`Consequently, in 4 cycles (stages S0-S3 and assuming no stalling), state S(t+T) will finally be computed.`" Final Action, ¶20(b). It is not sufficient that a future state will finally be computed by single (fourth) stage of the pipeline, rather the claim requires grouping logic pipelined to compute over plural cycles, T, a future state, S(t+T), of the processor based on a prior state, S(t), of the processor.

Sites does not disclose or suggest the required computation over plural cycles of a future state, S(t+T), of the processor based on a prior state, S(t). Accordingly, claim 17 and those dependent therefrom (claims 18-25) are all allowable.

Claim 18

With regard to claim 18, Applicant respectfully submits that no operation of *Sites* properly construable as "`intra-group dependency checking`" spans at least two execution cycles, as claimed. Through claim 18 is of substantially different scope that claim 13 (and is therefore argued separately), the Office's interpretation of *Sites'* first two pipeline stages, i.e., fetch (S0) and swap (S1), as intra-group dependency checking that spans multiple cycles is deficient for reasons similar to those given above with respect to claim 13. In short, attribution of dependency checking to stage S0 is simply unsupportable on its face.

As a preliminary matter, Applicant notes that (as previously argued above), *Sites* swap stage (S1) does not constitute intra-group dependency checking. However, even assuming

arguendo the Office's faulty premise, it is clear and unambiguous that the preceding stage (fetch stage S0) has nothing at all to do with dependency checking.

Nonetheless, the Office construes *Sites'* first two pipeline stages, i.e., fetch (S0) and swap (S1), as intra-group dependency checking that spans multiple cycles. *See* Final Action, ¶¶16, 21. To be specific, the Office states that: "`fetching is considered a portion of intra-group dependency checking because if instructions aren't fetched, then they cannot be checked for dependencies.`" Without acquiescing in the faulty premise that swap stage (S1) constitutes intra-group dependency checking, Applicant notes that the Office's attribution of dependency checking to the preceding stage S0 is simply unsupportable on its face. Applicant respectfully requests that the rejection be withdrawn. In addition to the reasons previously given, claim 18 is allowable for at least the foregoing reason as well.

Additional Consideration with Respect to Claims 16, 20, 30, 31, 33 and 34

Finally, present rejections of claims reciting the evaluation or checking of "non-deterministic conditions" are flawed because of the errors (described above) in the Office's extrapolation of a definition of *non-determinism*. Properly interpreted, neither the express nor inherent description of *Sites* discloses or suggests, evaluation or checking of non-deterministic conditions as recited in the claims. While claims 16, 20, 21, 30, 33 and 34 are allowable at least for the reasons previously given with respect to claims from which they depend, Applicant has also specifically addressed (above) the Office's errant definition and does not acquiesce in the Office's interpretation of claim limitations involving "non-deterministic condition" checks as an effective nullity.

Applicant has previously pointed out the Office's errant extrapolation of a definition for "non-determinism" and has provided a more correct and substantive definition in the response filed 7 February 2005. That more correct and substantive definition is repeated above.

CONCLUSION

For at least the foregoing reasons, Appellant's presently claimed invention(s) are not anticipated by the express or inherent disclosure of the cited prior art. Accordingly, this

honorable Board is respectfully requested to reverse the rejections of claims 9-34 and to direct the claims of the present application to be issued.

Respectfully submitted,

David W. O'Brien, Reg. No. 40,107
Attorney for Applicant(s)
(512) 338-6314
(512) 338-6301 (fax)

CERTIFICATE OF MAILING OR TRANSMISSION

I hereby certify that, on the date shown below, this correspondence is being

☒ deposited with the US Postal Service with sufficient postage as first class mail and addressed as shown above.

☐ facsimile transmitted to the US Patent and Trademark Office.

David W. O'Brien — 6-JUN-05 (Date)

EXPRESS MAIL LABEL: _____

CLAIMS APPENDIX

1-8. (cancelled)

9. (previously presented) A superscalar processor that, for a given instruction instance, performs, over plural execution cycles of the superscalar processor, instruction grouping for dispatch, including both intra-group and inter-group dependency checking, wherein the instruction grouping for dispatch takes the plural execution cycles to complete.

10. (previously presented) The superscalar processor of claim 9, further comprising:

grouping logic implementing plural early pipeline stages of the superscalar processor; and

plural execution units of varying pipeline depth coupled to receive instructions dispatched from the grouping logic.

11. (previously presented) The superscalar processor of claim 9,

wherein the instruction grouping identifies successive groups of instructions from an instruction stream for dispatch to respective ones of plural execution units of the superscalar processor.

12. (previously presented) The superscalar processor of claim 11,

wherein the superscalar processor dispatches all instructions from a particular one of the successive groups before dispatching any instructions from a subsequent one of the successive groups.

13. (previously presented) The superscalar processor of claim 9,

wherein the intra-group dependency checking spans at least two of the plural execution cycles.

14. (previously presented) The superscalar processor of claim 9,

wherein the intra-group dependency checking is independent of the inter-group dependency checking.

15. (previously presented) The superscalar processor of claim 9,
wherein data dependency and resource allocation checks in earlier pipeline stages of instruction grouping are based, at least in part, on a predicted subsequent state of the superscalar processor.

16. (previously presented) The superscalar processor of claim 9,
wherein non-deterministic conditions are evaluated in a final stage of instruction grouping prior to dispatch.

17. (previously presented) A processor comprising:
plural functional units that execute instructions in respective numbers of processor cycles; and
grouping logic coupled to the functional units and pipelined to compute, over plural cycles, T, of the processor, a future state, S(t+T), of the processor based on a prior state, S(t), of the processor and based thereon to select a group of instructions from a program sequence thereof for dispatch to the functional units,
wherein the future state computing takes the plural cycles to complete.

18. (previously presented) The processor of claim 17, further comprising:
wherein intra-group dependency checking by the pipelined grouping logic spans two or more of the T cycles.

19. (previously presented) The processor of claim 17, further comprising:
wherein inter-group dependency checking is performed independently of intra-group dependency checking.

20. (previously presented) The processor of claim 17,
wherein intra-group dependencies are checked by the pipelined grouping logic beginning in a first of the T cycles; and
wherein non-deterministic dependency conditions are checked during a last of the T cycles.

21. (previously presented) The processor of claim 20,
wherein inter-group dependencies are checked independent of the intra-group dependencies.

22. (previously presented) The processor of claim 17,
wherein the grouping logic implements T stages of a pipeline of the processor.

23. (previously presented) The processor of claim 17,
wherein T is four.

24. (previously presented) The processor of claim 17,
wherein the functional units include at least one functional unit capable of receiving and completing an instruction for each of the processor cycles.

25. (previously presented) The processor of claim 17,
wherein the functional units include at least one functional unit requiring multiple of the processor cycles for receiving and completing an instruction.

26. (previously presented) A method of operating a processor, the method comprising:
identifying successive groups of instructions for dispatch to respective ones of plural execution units of the processor;
performing, during plural pipelined execution cycles of the processor, dependency checking amongst instructions of a later one of the groups and between the instructions of the later group and instructions of a preceding one of the groups; and
dispatching instructions of the later group only after all instructions of the preceding group have been dispatched,
wherein the performed dependency checking takes the plural pipelined execution cycles to complete.

27. (previously presented) The method of claim 26,
wherein the dependency checking is performed by pipelined grouping logic.

28. (previously presented) The method of claim 26,

wherein intra-group dependency checking and within-group resource allocation are performed in successive processor cycles by pipelined grouping logic.

29. (previously presented) The method of claim 26,

wherein intra-group dependency checking is performed independent of inter-group dependency checking.

30. (previously presented) The method of claim 26,

wherein non-deterministic conditions are evaluated in a final one of the pipelined execution cycles implemented by pipelined grouping logic.

31. (previously presented) The method of grouping instructions for dispatch to execution units of a processor, the method comprising:

in a first cycle of processor execution, identifying plural candidate instructions for an instruction group;

in a subsequent cycle of processor execution, beginning intra-group dependency checking as amongst instructions of the instruction group;

in a cycle of processor execution prior to dispatch of any instruction from the instruction group, checking non-deterministic conditions; and

in a cycle of processor execution prior to the non-deterministic dependency condition checking, initiating inter-group dependency checking between instructions of the instruction group and instructions of one or more prior instruction groups,

wherein, for instruction instances of the instruction group, dependency checking, which includes the intra-group dependency checking and the inter-group dependency checking, takes plural of the cycles of processor execution to complete.

32. (previously presented) The method of claim 31, further comprising:

dispatching one or more instructions of the instruction group only after all instructions of the one or more prior instruction groups have been dispatched.

33. (previously presented) The method of claim 31,

wherein the non-deterministic condition checking is performed conservatively, based on an assumption of no change in condition.

34. (previously presented) The method of claim 31,

wherein the non-deterministic condition checking is performed aggressively, computing dispatch conditions for at least two alternatives including no change in condition and condition resolution; and

wherein a control signal is selective for a particular one of the computed dispatch conditions.

35. (previously presented) An apparatus comprising:

plural functional units; and

means for grouping, over plural pipeline stages, instructions for dispatch to respective ones of the functional units, wherein the grouping of a particular set of instruction instances takes the plural pipeline stages to complete.

36. (previously presented) The apparatus of claim 35, wherein the grouping means includes:

means for deriving over T processor cycles, a future state S(t+T) based on a present state; and

means for determining a group of the instructions to dispatch at time t+T.

EVIDENCE APPENDIX

There is no evidence submitted pursuant to 37 C.F.R. § 1.130, 1.131, or 1.132 or any other evidence entered by the examiner and relied upon by appellant in the appeal.

RELATED APPEALS APPENDIX

There are no decisions rendered by a court or the Board in any proceeding identified above in the Related Appeals and Interferences section.